



Corso NoSQL

settembre 2023





NoSQL

Cosa è NoSQL?

NoSQL vs SQL

4 tipi di database

Descrizione, esempi ed esercizi



Cosa è NoSQL



NoSQL

Approccio alla gestione dei database

Flessibile



KEY-VALUE



DOCUMENT



TABULAR



GRAPH



NoSQL

NON-relazionali

Distribuiti

Scalabili

Partition tolerant

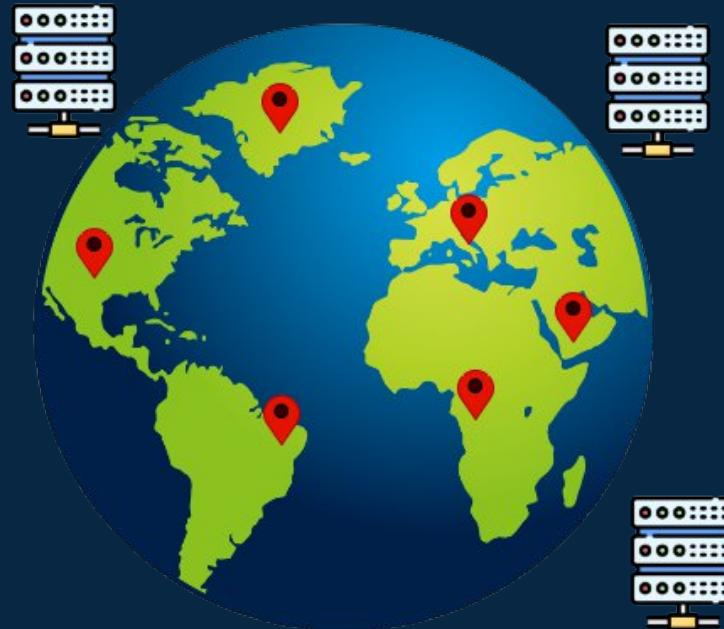
Alta disponibilità



NoSQL

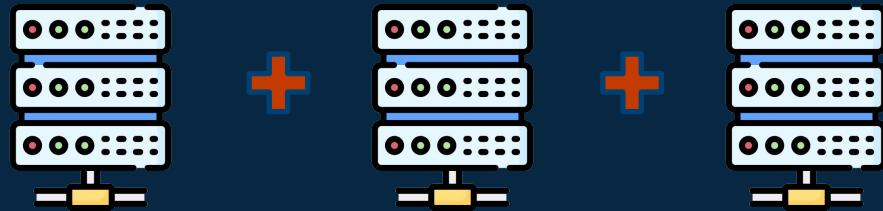
Distribuiti

Il software di gestione del database è installato su più servers (**nodi**).
I dati sono divisi fra vari nodi in base ad una **chiave di partizione**.
I nodi comunicano fra loro in rete; dal punto di vista del client, appaiono come un **unico sistema**.





NoSQL



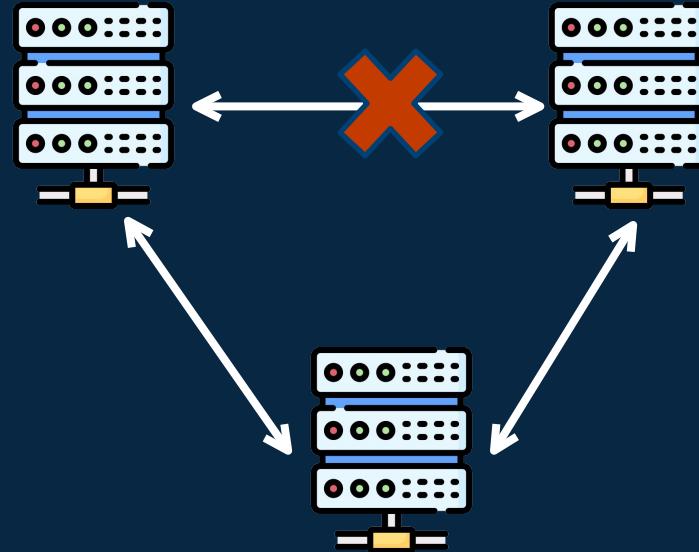
Scalabili

Hanno la capacità di memorizzare e recuperare dati su [scala internet](#) con prestazioni elevate, grazie alla scalabilità orizzontale ([scale out](#)). Se si ha bisogno di più spazio di storage o potenza di calcolo, si aggiungono altri nodi.



NoSQL

Partition tolerant

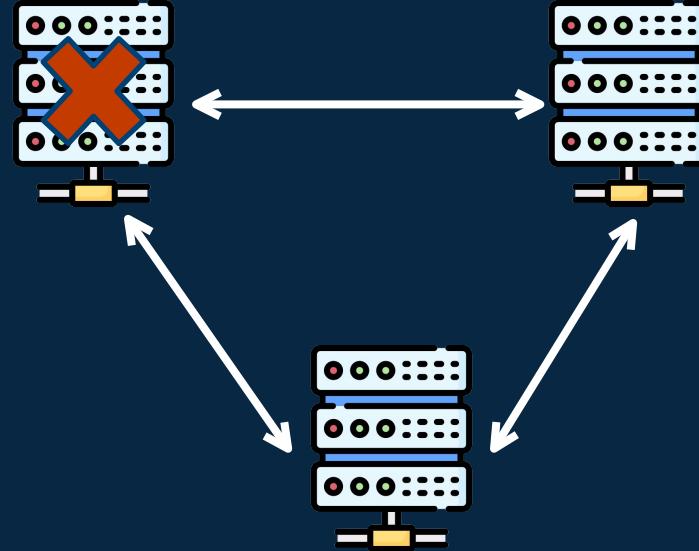


Sono in grado di lavorare in presenza di partizioni di rete ([network partitioning](#)).



NoSQL

Alta disponibilità



Sono in grado di servire una richiesta anche nel caso in cui un server sia down, grazie ai meccanismi interni di replica dei dati.



NoSQL

NON-relazionali

Flessibile

NON relazionali significa che non sono vincolati ai principi e alle tecnologie utilizzati nei database relazionali. Possono usare SQL o no, possono utilizzare le tabelle con struttura fissa o no, offrono parziale supporto per le transazioni ACID, ecc.

Relazionali

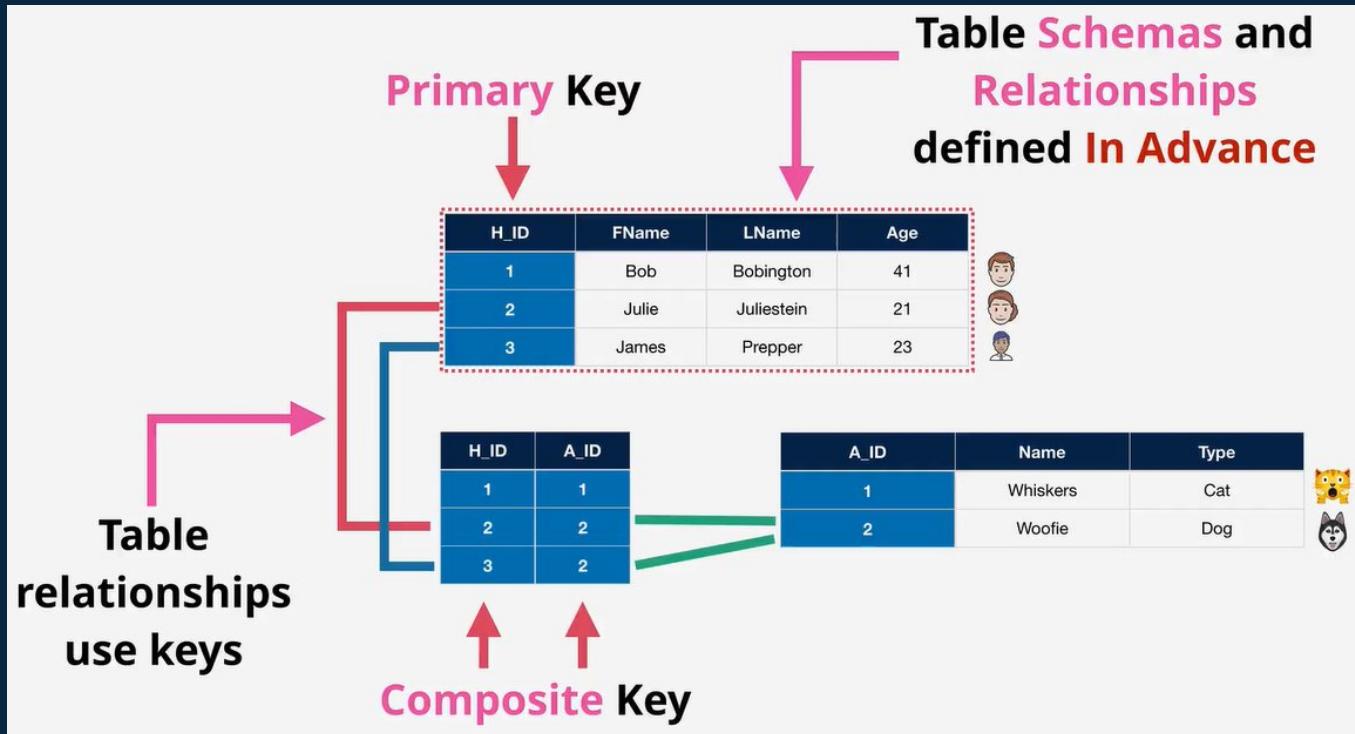
Rigido

- Sono progettati per essere eseguiti principalmente su un solo server
- Scalabilità verticale ([scale up](#))
- Modello di consistenza [ACID](#)
- Memorizzano i dati in tabelle con struttura fissa
- Utilizzano il linguaggio [SQL](#)

NoSQL = Not only SQL

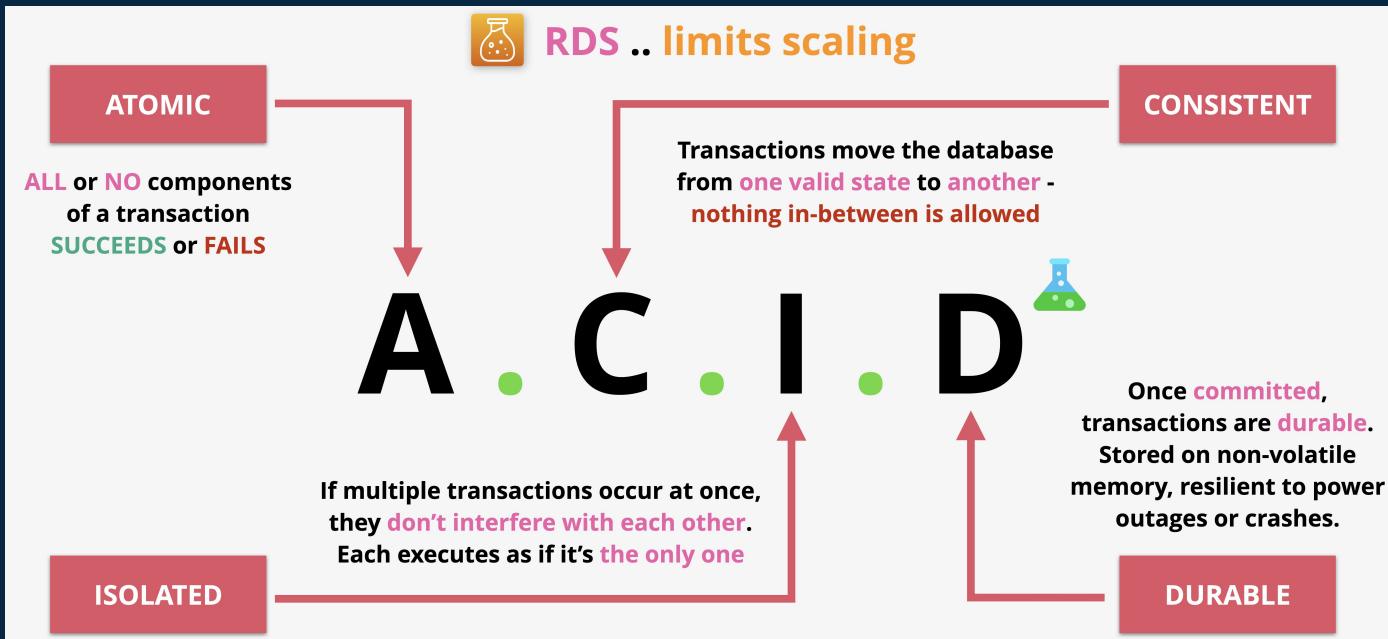


Database Relazionali: recap





Database Relazionali: recap





Database Relazionali: recap

1FN

- ✓ Ogni riga deve essere identificata univocamente
- ✓ Ogni colonna contiene un solo valore (no liste)
- ✓ Ogni valore non deve essere divisibile



Database Relazionali: recap

2FN

- ✓ La tabella è in 1FN
- ✓ Non sono presenti dipendenze parziali

Student ID	Course ID	Course Fee
Composite Key		
1	1	500
1	2	1000
2	4	200
2	3	750
3	5	1000
3	3	750



Student Courses	
Student ID	Course ID
Composite Key	
1	1
1	2
2	1
2	3
3	5
3	3

Course Fees	
Course ID	Course Fee
1	500
2	1000
3	750
4	200
5	1000
6	300





Database Relazionali: recap

3FN

- ✓ La tabella è in 2FN
- ✓ Non sono presenti dipendenze transitive

Composite Key			
Tournament Name	Year	Winner	Winner's DOB
Indiana Invitational	1998	Al Fredrickson	21 July 1975
Cleveland Open	1999	Bob Albertson	28 September 1968
Des Moines Masters	1999	Al Fredrickson	21 July 1975
Indiana Invitational	1999	Chip Masterson	14 March 1977

Tournament Winners		
Composite Key		
Tournament	Year	Winner
Indiana Invitational	1998	Al Fredrickson
Cleveland Open	1999	Bob Albertson
Des Moines Masters	1999	Al Fredrickson
Indiana Invitational	1999	Chip Masterson

Winners DOBs	
Winner	Date of birth
Chip Masterson	14 March 1977
Al Fredrickson	21 July 1975
Bob Albertson	28 September 1968



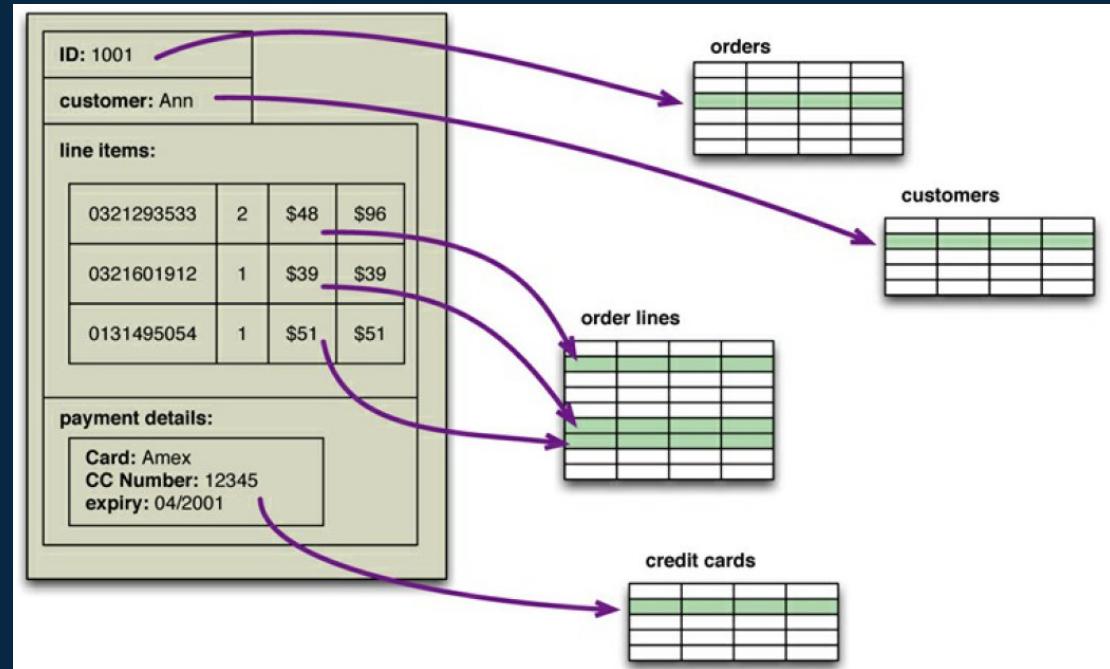


Perché usare NoSQL



Produttività sviluppo applicazioni

NoSQL può fornire un modello dati che soddisfa meglio le necessità dell'applicazione, alleviando il problema cosiddetto **impedance mismatch**.



Impedance mismatch



Dati su larga scala

Le applicazioni odierne raccolgono una quantità enorme di dati che non può essere più gestita dalla scalabilità verticale fornita dai database relazionali.

La scalabilità verticale prevede di aggiungere più CPU, disco e memoria ad una singola macchina, ma più risorse si aggiungono più l'operazione diventa costosa.

Inoltre vi è un limite fisico alle dimensioni che una singola macchina può raggiungere.

L'alternativa offerta da NoSQL è la scalabilità orizzontale.

I database NoSQL sono progettati per essere eseguiti su un gruppo ([cluster](#)) di macchine più piccole che utilizzano hardware standard ed intercambiabile ([commodity hardware](#)) per abbattere i costi.

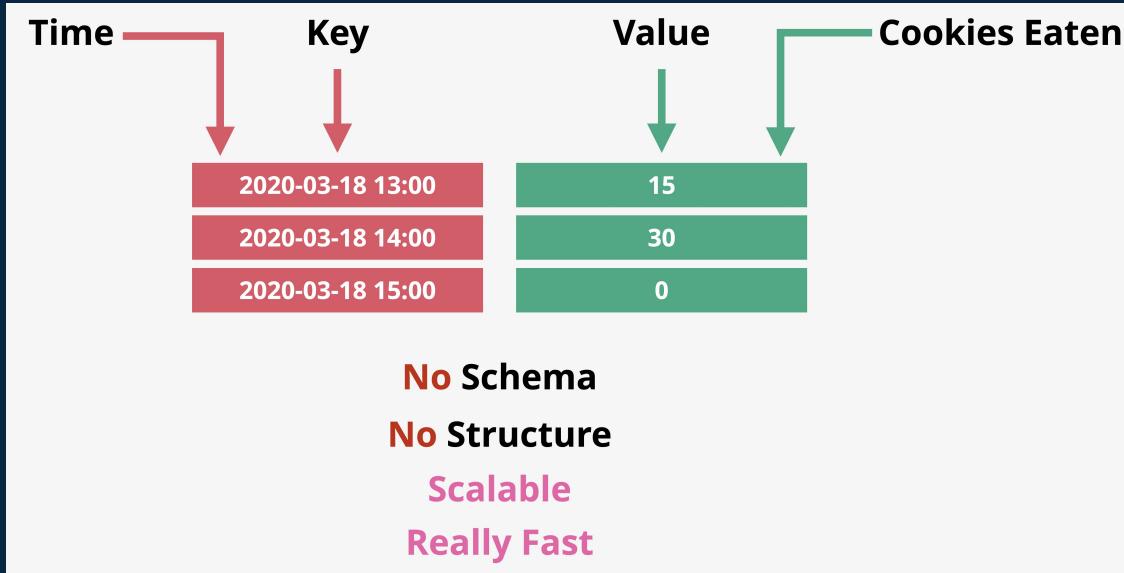
Inoltre il sistema risulta più robusto poiché il failure di una singola macchina non compromette il funzionamento dell' intero cluster.



Tipi di database NoSQL



Key-Value



Key-Value

- Lista di coppie chiave-valore
- La chiave è importante poiché identifica univocamente le coppie ed è usata per scrivere e interrogare i dati. Non è possibile eseguire query utilizzando campi non chiave
- Il valore può essere qualsiasi cosa: testo, json, xml, immagini...
- Scalabile poiché i dati possono essere divisi su più servers

Scenari

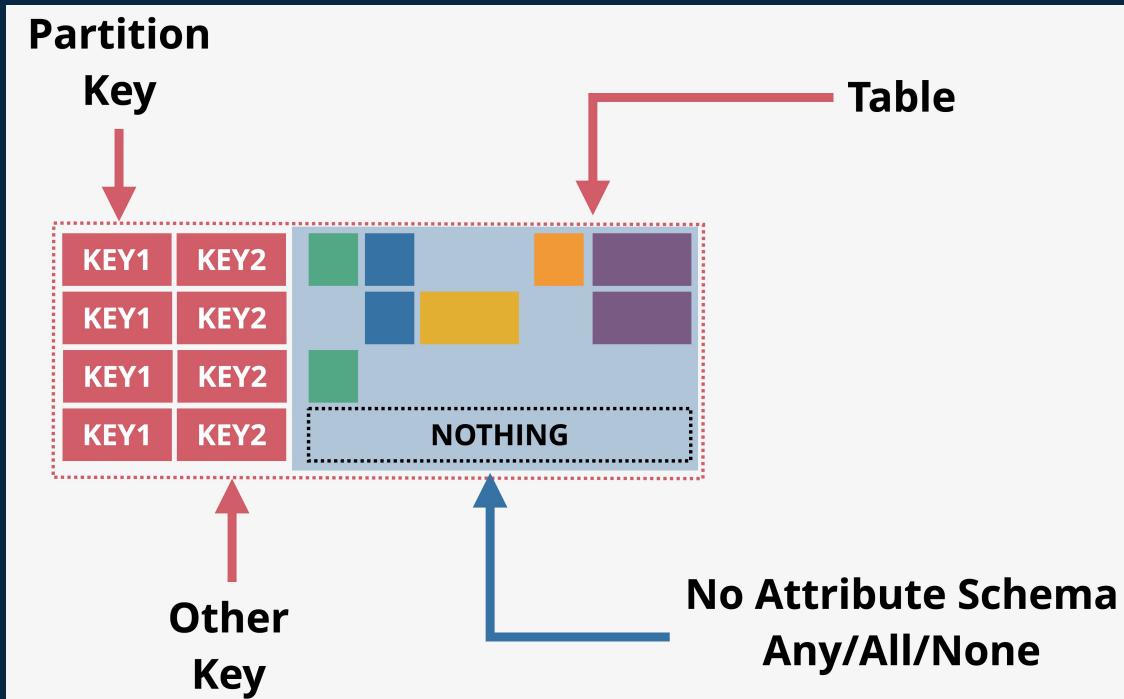
- Requisiti semplici
- Dati rappresentati come nome-valore o chiave-valore
- Dati non strutturati, nessuno schema
- In memory caching

Esempi

- Redis
- Memcached



Tabular o Wide Column



Wide-Column

- Evoluzione del modello precedente
- La chiave è importante poiché identifica univocamente gli items ed è usata per scrivere e interrogare i dati.
- Non è possibile eseguire query utilizzando campi non chiave
- la chiave può essere composta
- La struttura della chiave è comune a tutti gli items
- Scalabile poiché i dati possono essere divisi su più servers

Scenari

- Scalabilità a livello web
- Velocità recupero dati
- Non occorrono query ad hoc

Esempi

- DynamoDB
- Cassandra



Document

Documents

```
{  
    "_id": "1337",  
    "date": "10/10/2017",  
    "ship_status": "dispatched",  
    "orderitems": [  
        {  
            "itemid": "123",  
            "price": 9999.99  
        },  
        {  
            "itemid": "321",  
            "price": 0.99  
        }]  
}
```

Orders

```
<contact>  
    <id>1337</id>  
    <firstname>Bob</firstname>  
    <lastname>Bobington</lastname>  
    <phone type="Cell">(123) 456-7890</phone>  
    <phone type="Work">(890) 123-1337</phone>  
    <address>  
        <type>Home</type>  
        <street1>29 Acacia Road</street1>  
        <city>Nuttytown</city>  
        <country>UK</country>  
    </address>  
</contact>
```

Contacts

Ideal Scenarios : Interacting with **whole documents** or **deep attribute interactions**

Document

- I dati sono rappresentati come documenti
- Ogni documento è identificato da un **ID** e ha la propria struttura
- Si può interagire con l'intero documento o eseguire ricerche avanzate utilizzando campi annidati nel documento

Scenari

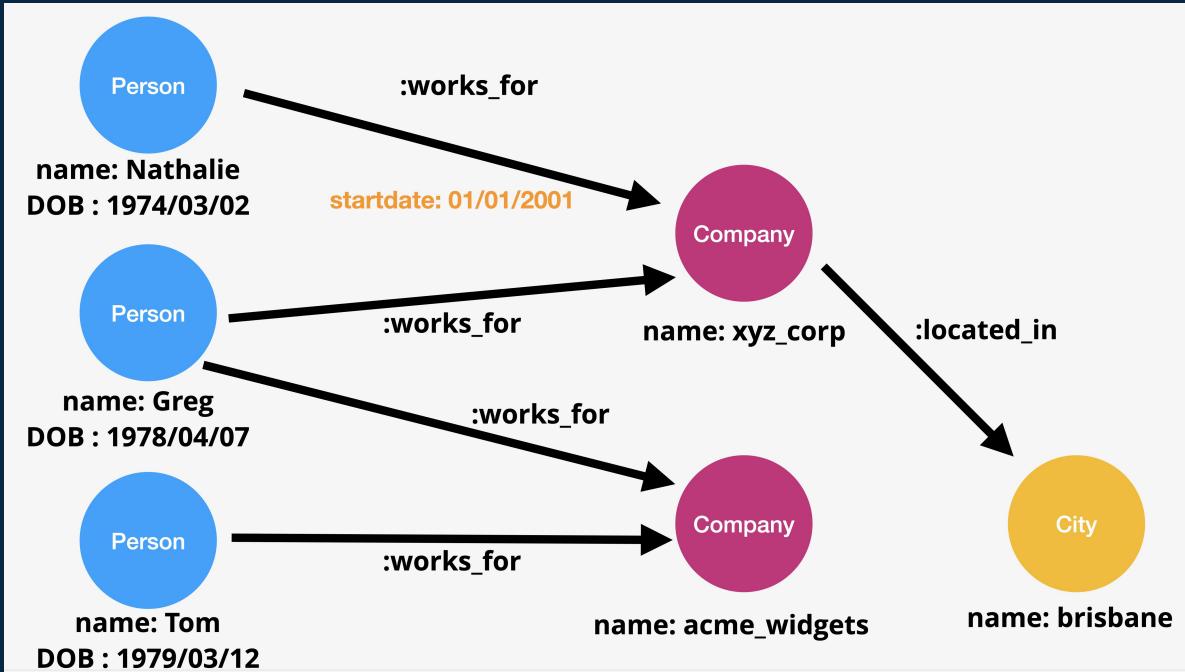
- Cataloghi
- Profili utenti
- CMS con gerarchie di documenti e/o links fra i documenti

Esempi

- MongoDB



Graph



Graph

- Ottimizzato per gestire relazioni complesse
- Le relazioni sono definite e memorizzate nel database assieme ai dati; non vengono calcolate ogni volta che si esegue una query, come avviene nei database relazionali
- Nodi: **Person**, **Company**, **City**
- Relazioni: **works_for**, **located_in**
- Attributi: **name**, **DOB**, **startdate**

Scenari

- Relazioni complesse
- Social media

Esempi

- Neo4j

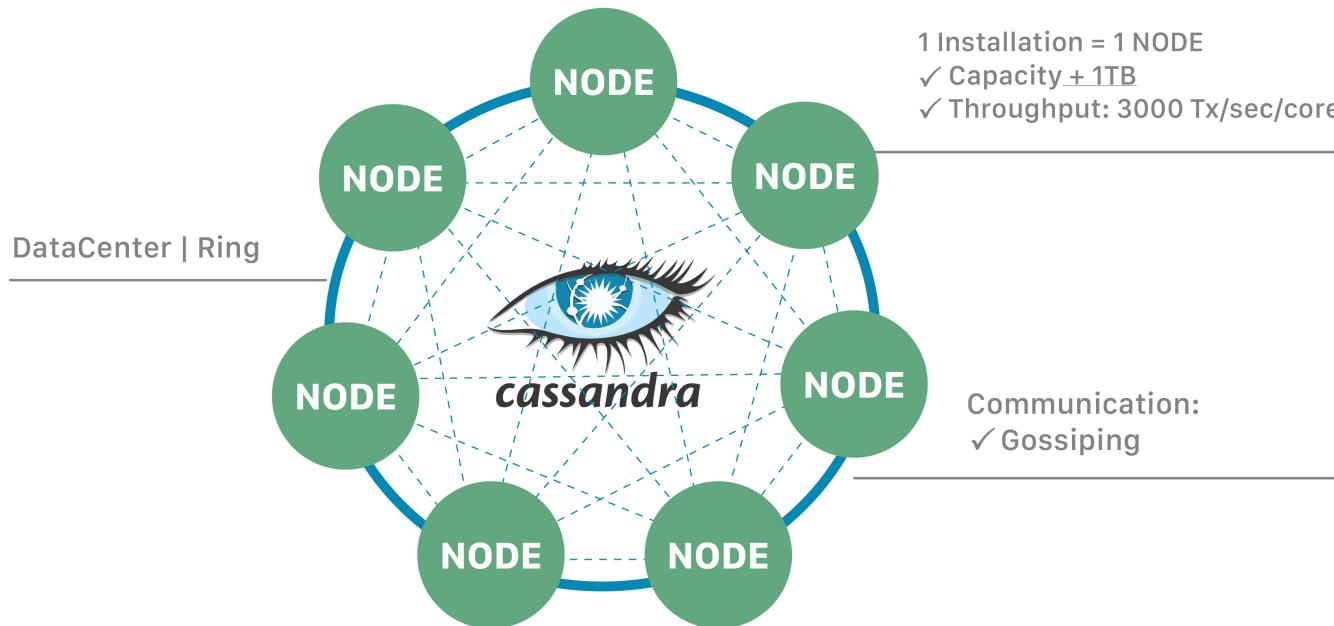


Cassandra



Cassandra - fondamenti

ApacheCassandra™ = NoSQL Distributed Database





Cassandra - fondamenti

Petabyte Database

Gestisce petabyte di dati mantenendo performance elevate

Alta disponibilità

Progettato per essere sempre disponibile anche in caso di failure dei nodi grazie alla sua natura distribuita e ai meccanismi built-in di replica dei dati. Modello master-less

Distribuzione geografica

Possibilità di creare più datacenter in varie regioni del mondo per avere un database globale. Cassandra gestisce automaticamente la comunicazione e la replica dei dati fra i datacenter

Performance

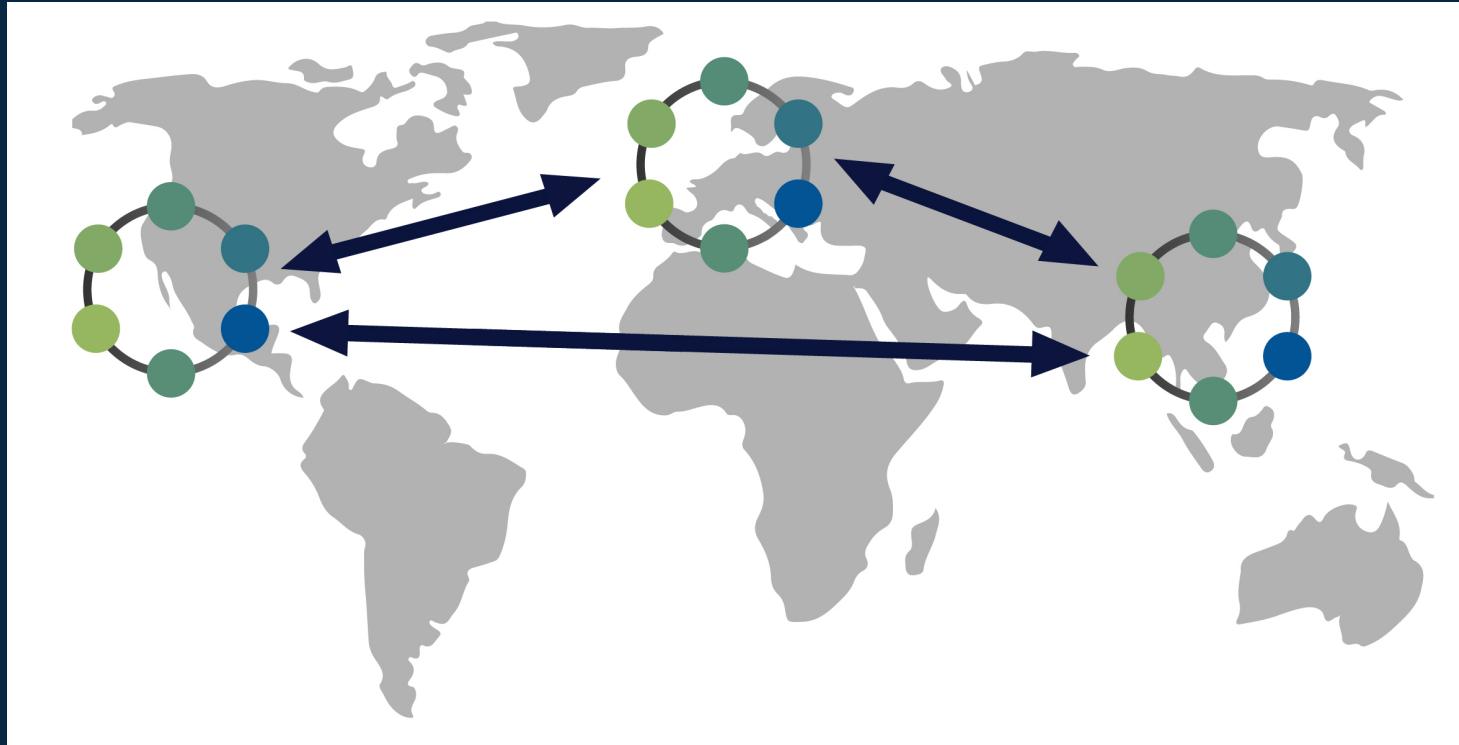
Performance consistenti su qualsiasi scala

Indipendente dal vendor

Può essere distribuito sui maggiori cloud provider o on premises



Cassandra - fondamenti





Cassandra - partizioni

The diagram illustrates the concept of partitions in a distributed system like Cassandra. It features a circular arrangement of nine orange nodes connected by black lines. In the center of this circle is a blue eye icon with the word "cassandra" written below it.

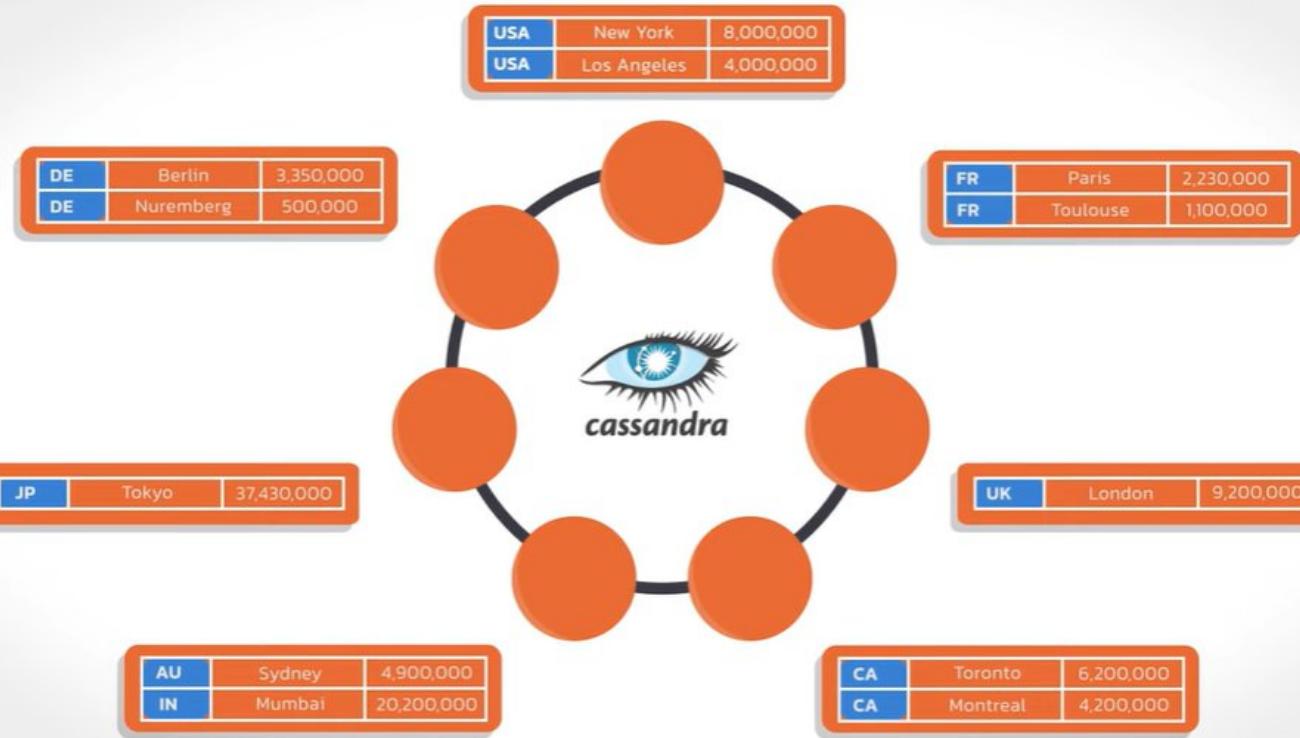
PARTITIONS

Country	City	Population
USA	New York	8,000,000
USA	Los Angeles	4,000,000
FR	Paris	2,230,000
DE	Berlin	3,350,000
UK	London	9,200,000
AU	Sydney	4,900,000
DE	Nuremberg	500,000
CA	Toronto	6,200,000
CA	Montreal	4,200,000
FR	Toulouse	1,100,000
JP	Tokyo	37,430,000
IN	Mumbai	20,200,000

Partition Key

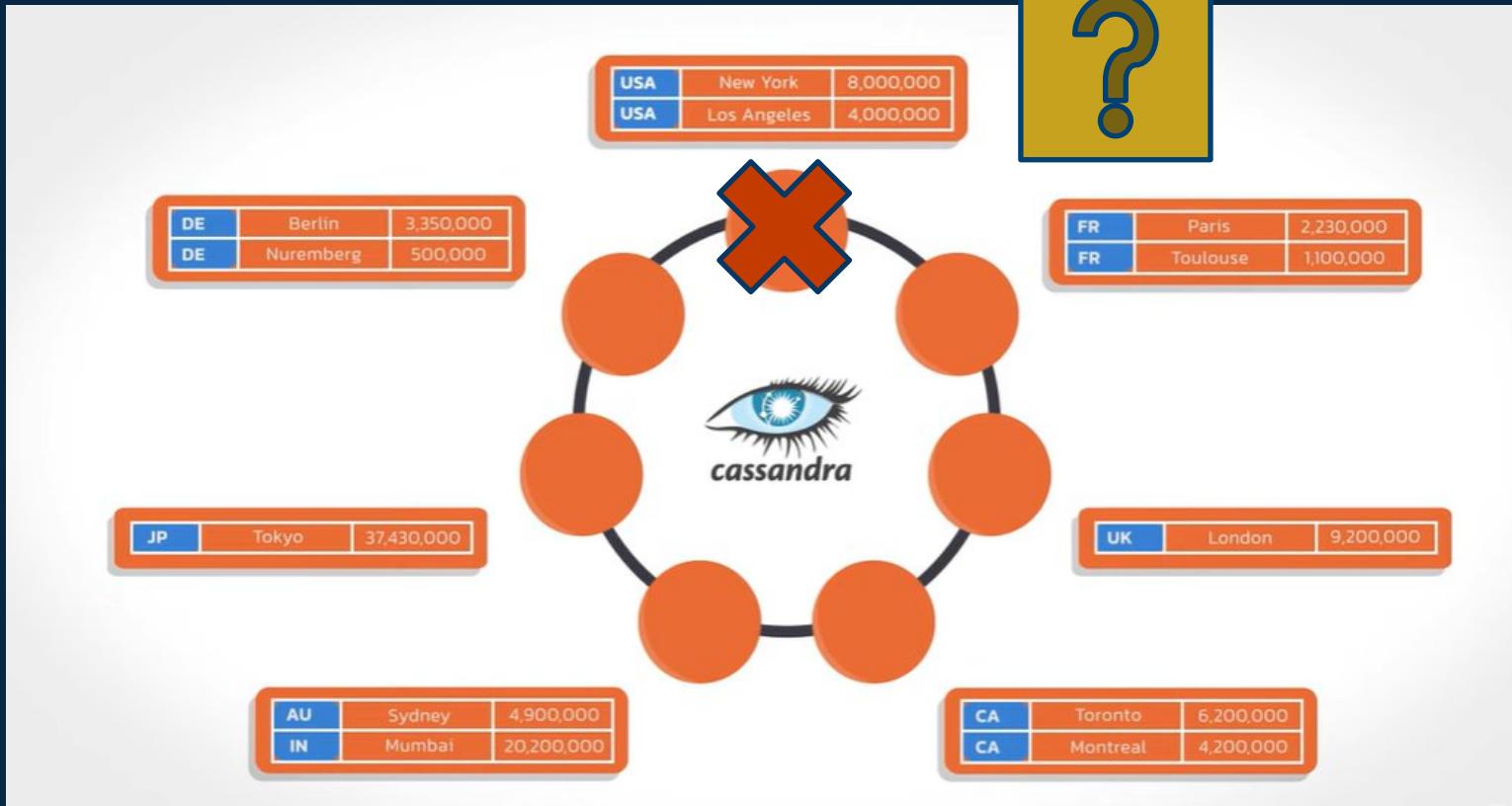


Cassandra - partizioni





Cassandra - partizioni





Cassandra - replica



- Ogni nodo ha un numero assegnato: **Partition Token**
- Cassandra utilizza una funzione hash per trasformare la chiave di partizione in un token e decidere su quale nodo salvare la riga. Ad esempio **HASH('USA')=59**
- Il parametro **Replication Factor (RF)** determina quante copie della riga devono essere salvate
- La figura accanto è relativa a **RF=1**
- Il nodo 0 memorizza i token nel range 84-0
- Il nodo 17 memorizza i token nel range 1-17
- Il nodo 33 memorizza i token nel range 18-33
- Il nodo 50 memorizza i token nel range 34-50
- Il nodo 67 memorizza i token nel range 51-67
- Il nodo 83 memorizza i token nel range 68-83



Cassandra - replica



- La figura accanto è relativa a **RF=2**
- Ogni nodo assume la responsabilità di un secondo range di tokens
- La copia è salvata sul nodo successivo (in senso orario) al nodo primario (nodo 67), in questo caso il nodo 83
- I colori nell'anello individuano un range; ad esempio il colore nero individua il range 84-0



Cassandra - replica



- La figura accanto è relativa a **RF=3**
- Ogni nodo assume la responsabilità di un terzo range di tokens
- La copia è salvata sui 2 nodi successivi (in senso orario) al nodo primario (nodo 67), in questo caso i nodi 83 e 0



Cassandra - coerenza

La coerenza indica quanto sono recenti e sincronizzate tutte le repliche di una riga. Con la replica dei dati nel sistema distribuito, ottenere la coerenza dei dati è un compito molto complicato.

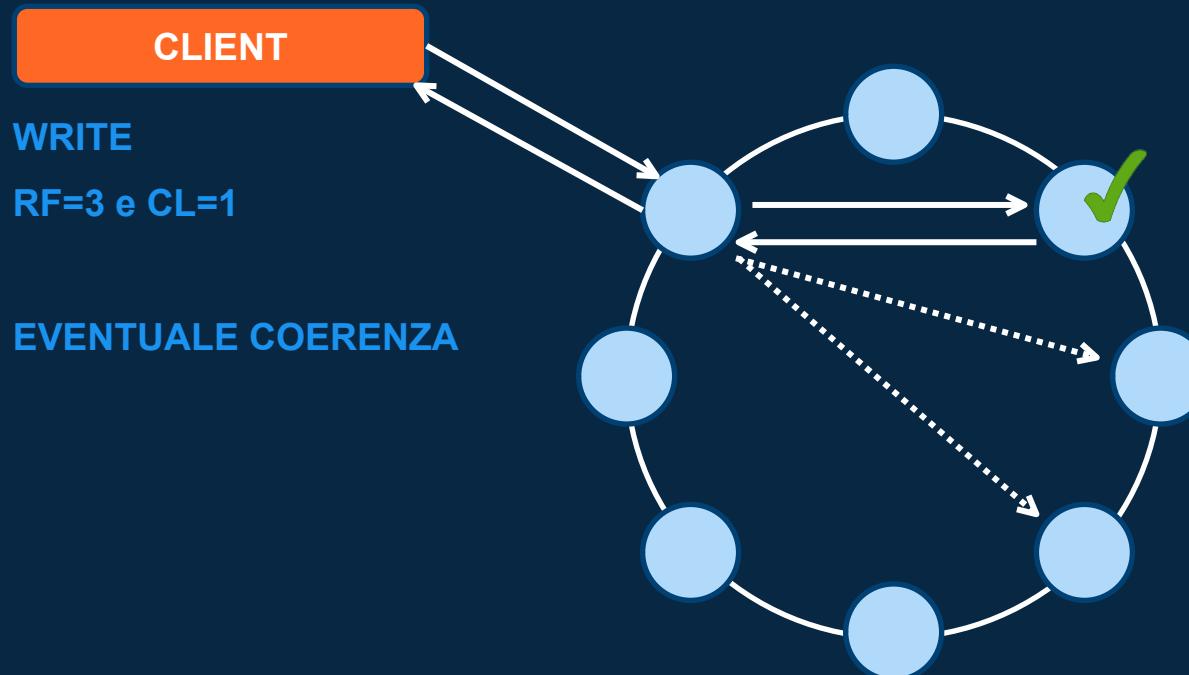
Cassandra come default sceglie la disponibilità sacrificando la coerenza ma offre la possibilità di aumentare il livello di coerenza per i casi d'uso che lo richiedono. Il parametro che permette di fare questo è: **consistency level (CL)**.

CL rappresenta il numero minimo di nodi che devono confermare l'operazione di lettura/scrittura, prima che l'operazione stessa venga considerata conclusa con successo e sia inviata la risposta al client. Tale parametro è in relazione col parametro già visto **RF**.

Ad esempio se CL=1 per le operazioni di scrittura e RF=3, un solo nodo deve confermare l'operazione e la replica sugli altri due nodi avviene in modo asincrono.

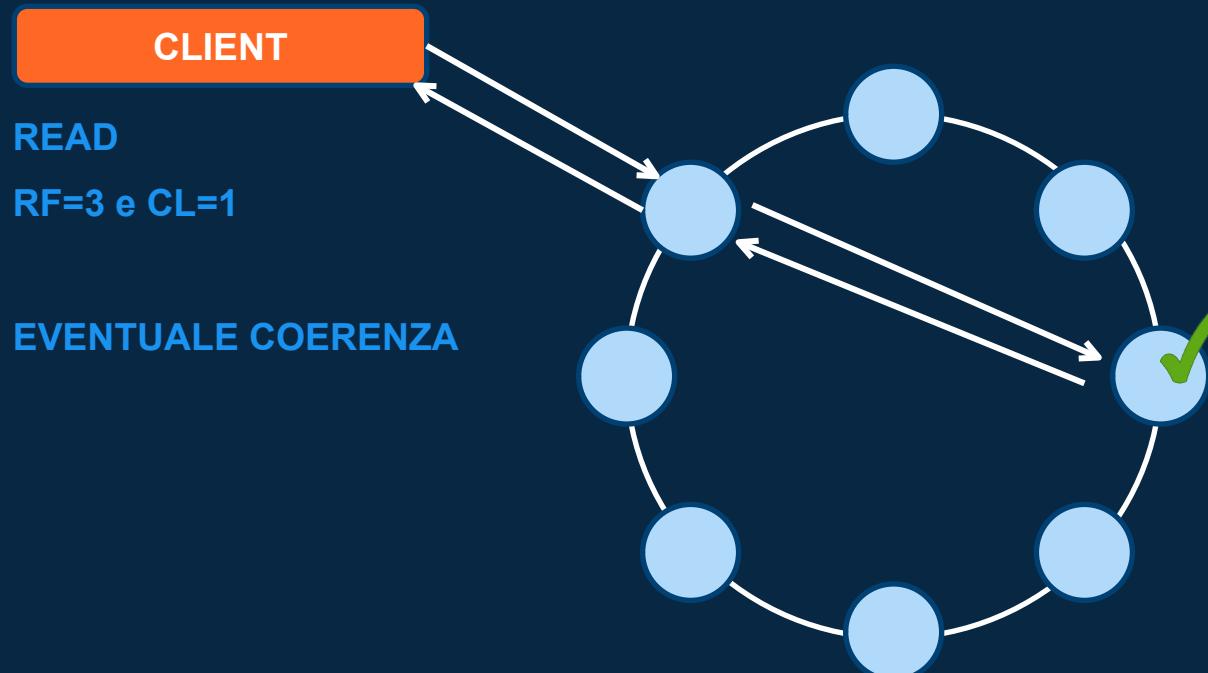


Cassandra - coerenza



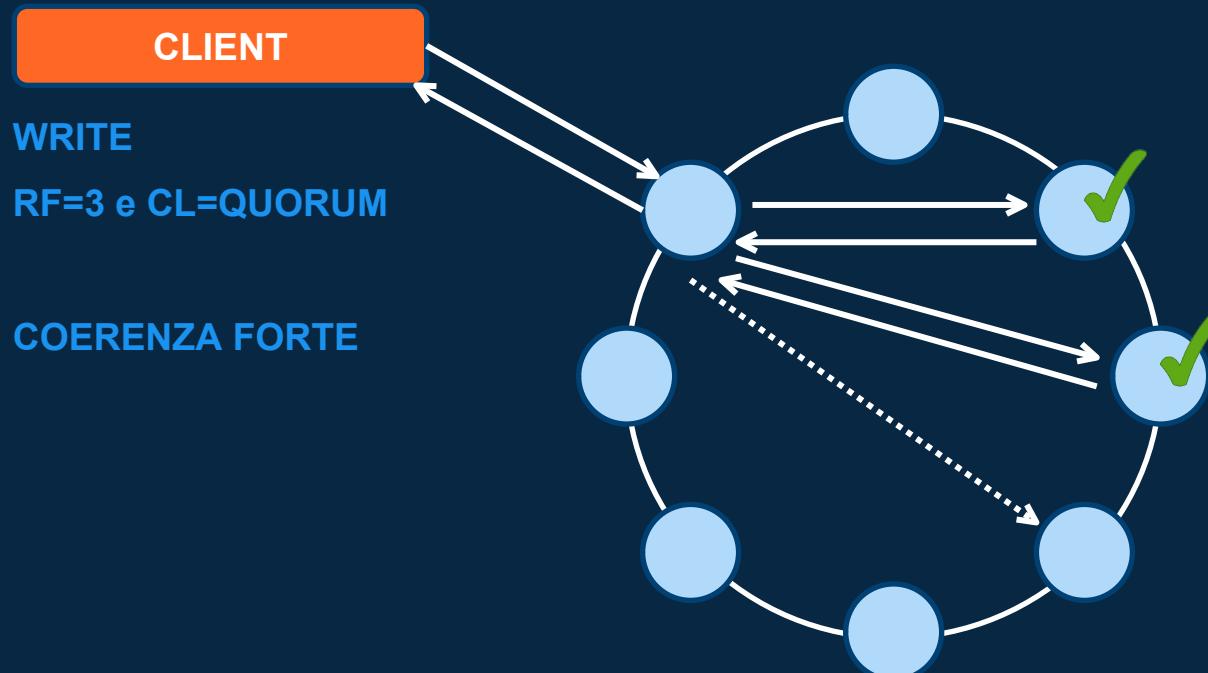


Cassandra - coerenza



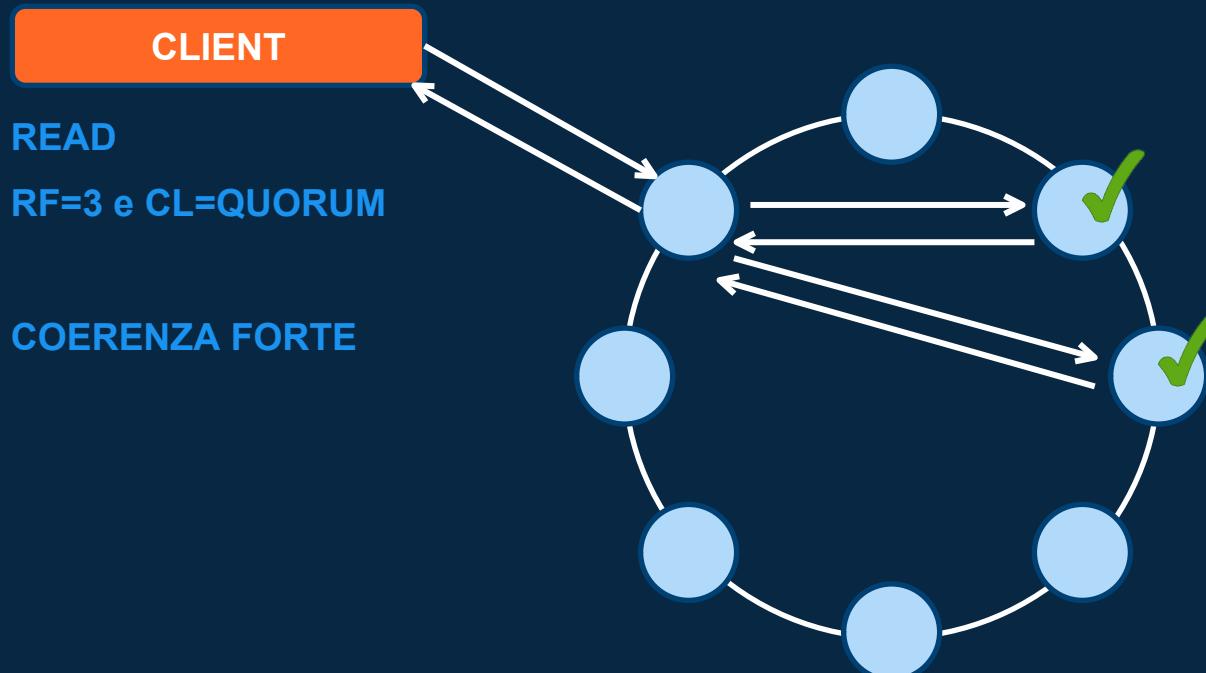


Cassandra - coerenza





Cassandra - coerenza



READ

RF=3 e CL=QUORUM

COERENZA FORTE

RF=3 e CL=QUORUM è un buon compromesso fra prestazioni, alta disponibilità e consistenza



Esercitazione





Riferimenti



Riferimenti

- ✓ NoSQL Database Tutorial <https://www.youtube.com/watch?v=xh4gy1lbL2k>
- ✓ Introduction to Cassandra <https://www.youtube.com/watch?v=jgqu1BcSKUI>
- ✓ What is Apache Cassandra? https://cassandra.apache.org/_cassandra-basics.html
- ✓ CAP theorem https://en.wikipedia.org/wiki/CAP_theorem
- ✓ Consistency Levels in Cassandra <https://www.baeldung.com/cassandra-consistency-levels>
- ✓ DataStax <https://astra.datastax.com>
- ✓ **Esercitazione:** <https://github.com/nmastrapasqua/corso-nosql>



Thank you

Tel
E-mail
Pec

080 4391385
info@sideagroup.com
sideagroup@pec.it

sideagroup.com